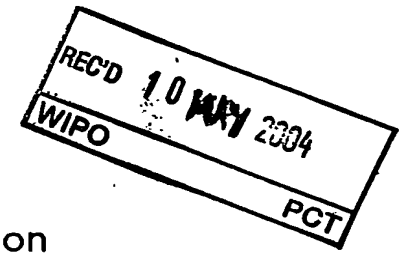
PCT/IB04/50479

Europäisches Patentamt | European Patent Office | Office européen des brevets




# Bescheinigung | Certificate | Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

**Patentanmeldung Nr. | Patent application No. | Demande de brevet n°**

03076189.4


Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.

R C van Dijk

Anmeldung Nr:
Application no.: 03076189.4
Demande no:

Anmeldetag:
Date of filing: 23.04.03
Date de dépôt:

Anmelder/Applicant(s)/Demandeur(s):

Koninklijke Philips Electronics N.V.
Groenewoudseweg 1
5621 BA Eindhoven
PAYS-BAS

Bezeichnung der Erfindung/Title of the invention/Titre de l'invention:
(Falls die Bezeichnung der Erfindung nicht angegeben ist, siehe Beschreibung.
If no title is shown please refer to the description.
Si aucun titre n'est indiqué se referer à la description.)

Method of and system to set an output quality of a media frame

In Anspruch genommene Priorität(en) / Priority(ies) claimed /Priorité(s) revendiquée(s)
Staat/Tag/Aktenzeichen/State/Date/File no./Pays/Date/Numéro de dépôt:

Internationale Patentklassifikation/International Patent Classification/ Classification internationale des brevets:

H04N17/00

Am Anmeldetag benannte Vertragstaaten/Contracting states designated at date of filing/Etats contractants désignées lors du dépôt:

AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HU IE IT LU MC NL PT RO SE SI SK TR LI

Method of and System to set an output quality of a media frame

We consider a scalable media processing algorithm that has been allocated a fixed CPU budget, i.e., it has been allocated a fixed share of the CPU power, on which it has to get by.

A a quality level control strategy is presented for a media processing algorithm with a fixed budget, based on Markov decision processes, that is used to
1. minimize the number of deadline misses,
2. minimize the number of quality level changes, and
3. maximize the quality level of processing.
To do so, it keeps track of a so-called relative progress and the previously used quality level. In addition to the method presented, we may apply so-called budget scaling, that can also handle long-term load fluctuations.

In the above method the Markov decision process is solved off-line, based on a number of typical video sequences. Next, the resulting control strategy is loaded into the run-time system (e.g. in the form of a table), to do the actual control. This table remains fixed during processing. A drawback of this method is that the video sequences that are encountered during run time may be different from the ones used to compute the control strategy. As a result, the control may be sub-optimal. Furthermore, a drawback is that the system depends on a table that has to be loaded, and which has to be computed off-line (i.e. the system is not self-contained).

To overcome the above problems, we propose a self-learning control strategy, for which we basically choose an on-line variant of the Markov decision process (it is based on reinforcement learning, or more specifically, on so-called Q-learning). In this way, an optimal control strategy is created on-line, tuned to the video sequences that are actually encountered. No control table needs to be loaded, but it is constructed

internally by the method itself. To make the Q-learning approach more efficient, we developed a method that extrapolates the effect of taking an action in a certain state to all possible actions in all possible states. In this way, the Q-learning approach learns drastically faster than usual.

Here, we discuss the techniques that underlie the self-learning quality control strategy. The control strategy is based on Reinforcement Learning (RL), and in particular on the techniques SARSA and Q-Learning [1].

Key in Reinforcement Learning are so-called (state,action)-values. We define the state of the application at a milestone as the combination of the scaled budget at the milestone, the relative progress at the milestone, and the quality at which the previous unit of work is processed. Per state $s$ and quality action $a$, a (state,action)-value $Q(s;a)$ is learned. This value reflects the expected reward in the long run when taking quality action $a$, starting from state $s$. At each milestone, if the application is in a state $s$, the quality action $a$ is chosen that has the largest value $Q(s;a)$. A problem is that we only can work with discrete states, while both the scaled budget and the relative progress are continuous variables. To solve this problem, we first define a discrete set of scaled budget points and a discrete set of relative progress points. Next, the idea is that only for the discrete gridpoints in the 3-dimensional (scaled budget, relative progress, previous quality) space, we keep track of the (state,action)-values. To approximate the (state,action)-value for a state that does not correspond to a gridpoint, we use linear interpolation.

In RL, after processing a unit of work, the (state,action)-value $Q(s;a)$ is updated that corresponds to the state $s$ prior to processing the unit of work. The quality action $a$ is the quality that is chosen to process the unit of work, i.e., the quality action $a$ for which $Q(s;a)$ was highest. The update is done by using the direct revenue of action $a$ in state $s$ and the (state,action)-value $Q(s';a')$ that corresponds to the state $s'$ that the application is in after processing the unit of work. Quality action $a'$ again is the quality action for which $Q(s';a')$ is highest.

Normally, in RL only the value $Q(s;a)$ is updated. However, we take a different approach by not updating only one (state,action)-value, but instead by updating the (state,action)-values at all gridpoints and for all possible quality actions. We do this as follows. After processing a unit of work in a certain quality, we first *estimate* the processing time of the unit of work in each other quality $a$ (see step 6 of the third algorithm). Next, for all gridpoints $s$, we determine the effect of taking all possible actions $a$ in state $s$, i.e., we determine the resulting state $s'$, and we use this in the traditional way to update $Q(s;a)$. We can do this, as we basically only need the (estimated) processing time for this (see steps 7-11 of the third algorithm).

For details, we refer to the following pseudo-code algorithms. In the code, *sbp* stands for scaled budget point, *rpp* stands for relative progress point, and *pq* stands for previously used quality. For information on budget scaling and [illegible] application factors, refer to [illegible].

Algorithm *initialize*

1. initialize the running complication factor
   $rcf \leftarrow 1$
2. for all states $(sbp, rpp, pq)$
3. for all quality actions $q$
4. initialize the (state,action)-value
   $Q(sbp, rp, pq; q) \leftarrow 0$

**Algorithm** *get decision quality*
**Input:** relative progress $rp$
**Input:** previously used quality $pq$
**Output:** decision quality $dq$

1. compute the scaled budget
   $sb \leftarrow b/rcf$
2. for scaled budget $sb$, relative progress $rp$, and previous quality $pq$, compute the interpolated (state,action)-values $Q_{ivec}(sb, rp, pq; q)$ for all possible quality actions $q$
3. decision quality $dq$ is the quality action $q$ that corresponds to the highest value $Q_{ivec}(sb, rp, pq; q)$

**Algorithm** *update (state,action)-values*
**Input:** processing time $pt$
**Input:** processing quality $q$

1. make a copy of the running complication factor corresponding to the situation that we had before processing the last unit of work
   $oldrcf \leftarrow rcf$
2. use $pt$ and $q$ to update the running complication factor
   $rcf \leftarrow rcf + \alpha \cdot (\frac{pt}{avg(q)} - rcf)$
3. compute the scaled budget
   $sb \leftarrow b/rcf$
4. for all states $(sbp, rpp, pq)$
5. for all quality actions $\tilde{q}$
6. estimate the processing time of the last unit of work for quality $\tilde{q}$
   $ept \leftarrow \frac{avg(\tilde{q})}{avg(q)} pt$
7. simulate processing the last unit of work in quality $\tilde{q}$, starting in state $(sbp, rpp, pq)$, and having a normalized processing time $ept/oldrcf$
8. observe both the resulting revenue $rev$ and the resulting relative progress $rp$
9. for scaled budget $sb$ (derived in line 3), relative progress $rp$, and previous quality $\tilde{q}$, compute the interpolated (state,action)-values $Q_{ivec}(sb, rp, \tilde{q}; \tilde{q}')$ for all possible quality actions $\tilde{q}'$
10. $Q'$ is the highest value $Q_{ivec}(sb, rp, \tilde{q}; \tilde{q}')$
11. update the (state,action)-value $Q(sbp, rpp, pq; \tilde{q})$ using $rev$ and $Q'$
    $Q(sbp, rpp, pq; \tilde{q}) = Q(sbp, rpp, pq; \tilde{q}) + \beta \cdot (rev + \gamma \cdot Q' - Q(sbp, rpp, pq; \tilde{q}))$

## References

[1] R.S. Sutton and A.G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

# Adaptive QoS Control for Real-Time Video Processing

Clemens C. Wüst, Liesbeth (E.F.M.) Steffens, Wim F.J. Verhaegh
Philips Research Laboratories
Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands
{clemens.wust,liesbeth.steffens,wim.verhaegh}@philips.com

## Abstract

*Video processing in software is characterized by highly fluctuating, content-dependent, processing times, as well as strict real-time requirements, but worst-case resource allocation is not cost-effective. We present an approach based on asynchronous, scalable (or imprecise), processing and QoS adaptation, which allows close-to-average resource allocation. We propose two strategies for controlling the QoS-level during run-time, one static, and one dynamic. We enhanced both strategies with a compensation for structural (non-stochastic) load fluctuation. Finally, we validate our approach by means of a simulation experiment.*

## 1 Introduction

Consumer terminals, such as set-top boxes and digital TV-sets, currently apply dedicated hardware components to process video. In the foreseeable future, programmable hardware is expected to take over. Programmable hardware has the advantage of being applicable to future coding standards, and it allows the user to enhance the functionality of the consumer terminal. Consequently, video processing will move from hardware to software as soon as Moore's law permits.

High quality video has stringent jitter requirements; deadline misses are considered errors that affect the mean time between failure. Every frame (or field in interlaced video) has to be available when the renderer needs it for timely display.

Software video processing is characterized by highly fluctuating resource requirements, due to content dependencies. For example, Figure 1 shows the decoding times for a sequence of MPEG-2 frames on a TriMedia 1300 (180MHz) processor [9]. In the figure we can identify both temporal load fluctuations, as, for instance, within frames 1–100 and 101–310, and structural load fluctuations. Structural load fluctuations are, amongst others, caused by the varying complexity of video scenes. Typically, there is a large gap between the worst-case and average-case decoding times.

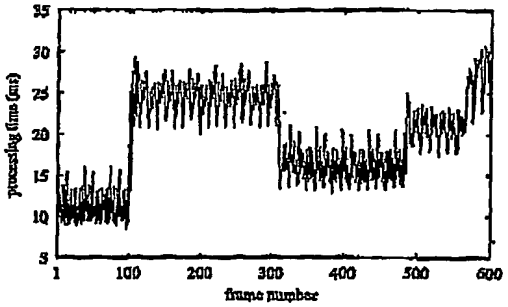



**Figure 1. The decoding times for a sequence of MPEG-2 frames, showing both temporal and structural load fluctuations.**

This leads to a dilemma. On the one hand, worst-case resource allocation is not cost-effective, since consumer terminals are heavily resource-constrained due to high pressure on cost. On the other hand, assigning resources closer to the average-case load situation may lead to deadline misses and/or disturb applications in other parts of the system.

We address this dilemma as follows. First, we provide temporal isolation between this task and other tasks in the system, by allocating a guaranteed periodic CPU budget to the task. This removes interference from other tasks an allows us to concentrate on this one task only. Second, we even out the CPU load by asynchronous video processing and working ahead: after processing a frame the task immediately can start processing the next frame, without waiting for the deadline. The extent to which working ahead can be applied is determined by latency and buffering limitations. Third, we use scalable video algorithms, which allow a trade-off between the processing time and the output quality [3]. These scalable video algorithms apply approximate

processing [6] to the video domain. With these ingredients in place, we optimize the Quality of Service (QoS) of the task, by controlling the processing quality for each individual frame. In this paper we describe a number of control algorithms that optimize QoS by balancing deadline misses, processing quality, and changes in the processing quality.

## 2 QoS RM Framework

The effort described in this paper is embedded in a larger effort, which defines and builds a framework for QoS resource management, called QoS RM [1, 7]. Maintaining the properties of robustness and cost-effectiveness are major requirements that are imposed on this framework. The framework consists of two main parts, a Resource Manager (RM) and a Quality Manager (QM), and is based on two main concepts, quality level and resource budget.

The QM assigns course-grained quality levels and resource budgets to so-called Resource Consuming Entities (RCEs), in a way that is very similar to the approach described in [4]. A course-grained quality level consists of a number of fine-grained quality levels, which roughly provide the same course-grain processing quality. During the steady state between two QM decisions, the RM provides guaranteed and enforced resource budgets, in a way that is very similar to the approach described in [5]. These resource budgets provide a virtual processor [1]: running on a fraction $\alpha$ of a processor with clock frequency $F$ is almost the same as running on a private processor with clock frequency $\alpha F$ [2]. An RCE controller within the RCE makes sure that the RCE produces an acceptable result, given the resource budget provided by the RM, by controlling the fine-grained quality level.

The work presented in this paper is aimed at RCEs in the context of a QoS RM framework, but it can also be applied outside this context. In the presence of a QM, the RCE controller can request a budget increase in case of a structural load increase. A stand-alone application cannot do this, and must therefore be able to address structural load fluctuations as well as temporal load fluctuations.

## 3 Basic Approach

We consider a single asynchronous scalable video processing task, with an associated controller. At each milestone, when a frame is completed, and a next frame is about to be processed, the controller decides what to do next. It may decide to skip a frame, because it has previously missed a deadline, but normally it simply selects a quality level for the next frame. Each frame must be completed before the corresponding deadline. Successive deadlines are strictly periodic, with a given period $P$. The processing time for a frame depends on both the chosen quality level and the content of the frame. During each deadline period, the task can use a guaranteed processing-time budget $b$.

When a deadline is missed, the renderer (outside the scope of this paper) redisplays the previous frame, which implies a QoS reduction for the stream. The controller detects a possible deadline miss after completion of the frame, at the milestone, and accounts for the redisplay by shifting the missed deadline and all subsequent deadlines by one period. If it has detected a deadline miss, the controller skips a suitable subsequent frame. Skipped frames are not taken into account.

In an asynchronous approach, it is important to know the relative progress at a milestone compared to the corresponding deadline, because a larger progress leads to a lower risk of missing the next deadline. Progress, measured at a milestone, is the total amount of guaranteed budget left until the corresponding deadline, divided by the periodic budget $b$. To obtain the progress, the controller monitors the resource allocation to the task. The progress is bounded from above due to finite buffer sizes, and is always non-negative because of the shifting of deadlines in case of deadline misses. Note that aborting the processing at a deadline miss would not make sense, it would only decrease the progress, and thus increase the risk of another deadline miss. Moreover, it is harder to abort a frame than to finish it.

### 3.1 Problem Statement

At each milestone the controller has to choose the quality level at which the upcoming frame is processed. The problem we consider is to choose this quality level in such a way that the following three objectives are met. First, because deadline misses result in severe artifacts in the output, they should be as sparse as possible. Second, to obtain a high output quality, frames should be processed at the highest possible quality level. Third, the number and size of quality level changes should be as low as possible, because (bigger) changes in the quality level may result in (better) perceivable artifacts. Clearly, meeting all three objectives at the same time is impossible, so we have to find an optimal balance.

### 3.2 Solution Approach

To address the above problem, we have modeled it as a Markov Decision Process (MDP) [8] and as a Reinforcement Learning (RL) problem [10]. Both methods are related, and consider a system that is characterized by a set of states, a set of possible actions for each state, and numerical revenues for state transitions. At discrete moments in time, the controller observes the state of the system, and takes an action. The action initiates a state transition, which depends

on the action itself and on the system's dynamics. On each action that is taken a revenue is obtained. The goal of both approaches is to maximize the average revenue over all state transitions.

For our problem, the state of the system is determined by the task's progress, and by its quality level, both at the current milestone. The action is the quality level for the next frame. A revenue is composed of a (high) penalty on the expected number of deadline misses before reaching the next milestone, a reward for choosing a particular quality level at the milestone, and a penalty for choosing a quality level different from the previously used one. In this way, we balance the three problem objectives. The obtained revenue can be viewed as an objective QoS measure.

The MDP is solved off-line for a particular processing budget, and requires pre-determined probabilities of state transitions. The result is given by a static policy, which is a set of state-action pairs. This policy is applied during run-time to choose the quality level at milestones, given the state. Clearly, the budget applied for processing should match the budget for which the policy is derived. We denote this control strategy by MDP. For more details, we refer to [11].

The RL approach can be considered as an on-line variant of the MDP. An RL control strategy starts with no knowledge at all, and has to learn optimal behavior from the experience it gains during run-time. This is achieved by constructing and maintaining a dynamic policy. To match our specific problem, we applied the SARSA algorithm [10]. This algorithm is based on learning state-action values, which estimate how good it is to choose a particular action in a given state. We took advantage of our specific problem to modify SARSA. This modification enables us to update all state-action values instead of just one at milestones, which speeds up convergence significantly. The state-action values are applied to choose the quality level for the next frame. Given the state, the quality level (= action) yielding the largest state-action value is chosen.

## 4 Structural Load Fluctuations

The MDP and RL control strategies assume successive processing times to be independent of each other. This is the case for temporal load fluctuations, but not for structural load fluctuations. In this section we compensate MDP and RL for this shortcoming. To track the structural load during run-time, we first compare the actual processing time *apt* of a completed frame with the expected processing time *ept* for a frame at the applied quality level. We express the relation as a complication factor *cf*, i.e., $cf = apt/ept$. The *ept* values are derived off-line. We filter out temporal load fluctuations by computing a running average *rcf* over the successive encountered *cf* values. If *rcf* deviates from one, then it feels to the task as if the processing budget deviates from the available budget $b$. In case of $rcf = 1.2$, a budget $b = 30$ ms would feel as a budget of only 25 ms, and in case of $rcf = 0.8$, that same budget would feel as a budget of 37.5 ms. We therefore define a *scaled budget* as $b/rcf$. During run-time, we compute the scaled budget at each milestone.

Given the scaled budgets, we modified the strategies MDP and RL as follows. In the MDP approach, we derive policies for a selected set of scaled budgets. During run-time, at each milestone, we approximate a policy for the current scaled budget by interpolating between two derived policies. In the RL-problem we include the scaled budget directly into the state space. We denote these adapted versions of MDP and RL by MDP* and RL*, respectively. For details on MDP*, we refer to [12].

## 5 Performance Results

To test the different control strategies, we have run a simulation experiment. In this experiment, we first measured the per-frame processing times for decoding a complete movie, consisting of 141,573 frames, using a scalable MPEG-2 decoder with four quality levels, $q_0, \ldots, q_3$, in order of increasing quality, running on a TriMedia platform [9]. We produced four traces, one for each quality level. With a frame rate of 25 frames/sec, we measured a worst-case processing time of 77.4 ms and an average-case processing time of 26.8 ms, for the highest quality level, $q3$. We used these traces to derive state-transition probabilities for the MDP, to determine *ept* values for budget scaling, and to simulate the processing under the different control strategies. In [12] we describe an experiment in which MDP* is applied on video sequences other than the one used to compute the state-transition probabilities. The performance of MDP* in the current experiment is roughly the same.

We set the upper bound on progress at 2, which implies that that the decoder can work ahead by at most two deadline periods. Based on discussions with experts in the video domain, we defined the revenues as follows. For each deadline miss we gave a penalty of 10,000. For choosing quality level $q_0, \ldots, q_3$ we gave rewards of 4, 6, 8, and 10, respectively. For increasing the quality by one level we gave a penalty of 10, for two levels we gave a penalty of 100, and for three levels we gave a penalty of 1000. For decreasing the quality level these penalties were multiplied by five.

For different control strategies, MDP, MDP*, RL*, Q3, we simulated processing the movie at different budgets. Q3 denotes the straightforward strategy of always choosing quality level $q_3$. Figure 2 shows the average revenue that we measured in these simulations, as a function of the budget. Using dynamic programming we computed the upper bound that can be obtained on the average revenue, which we denote by UB. This upper bound can not be met by

any control strategy, because it requires complete knowledge on all future processing times. Strategies MDP* and RL* perform close to optimum, and their graphs roughly overlap. Strategy MDP performs significantly worse, and strategy Q3 performs poorly. From this result we conclude that quality control and budget scaling are useful. It is interesting from a practical point of view to compare the budgets required to achieve a certain QoS value (average revenue). For example, to achieve an QoS value of 8, MDP* and RL* require a budget of 26.9 ms, MDP requires a budget of 30.0 ms, and Q3 requires a budget of 33.7 ms.

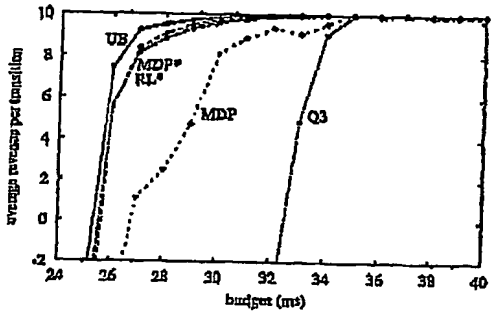



**Figure 2. The average revenue for different control strategies, as a function of the budget.**

Next, Figure 3 shows the average number of deadlines that were missed per frame. The graphs for UB, MDP, MDP* and RL* overlap, and Q3 performs significantly worse. At a budget of 26,9 ms, MDP* and RL* only face occasional deadline misses, while Q3 on average creates one deadline miss every 25 frames, which is once per second. Clearly, such a number of deadline misses is unacceptable in a consumer terminal.

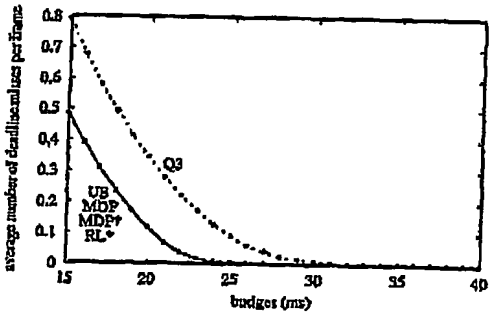



**Figure 3. The average number of deadline misses per frame for different control strategies, as a function of the budget.**

## 6 Conclusions and Ongoing Work

We have presented an approach to handle an average-case resource allocation, assigned to a software video processing task that is characterized by a highly fluctuating workload and strict real-time requirements. Our approach is based on asynchronous scalable processing, which allows a trade-off between the resource usage and an overall QoS level. Deriving proper QoS parameters to trade-off deadline misses and processing quality is a task of video domain experts. We introduced two strategies to control the QoS level during run-time. The MDP strategy is based on off-line optimization, whereas the RL strategy has to learn optimal behavior from run-time experience. We adapted MDP and RL to handle structural load fluctuations. The two resulting strategies, MDP* and RL*, proved to perform close to optimum in a simulation of processing an entire movie. Given an average-case resource allocation, MDP* and RL* met almost all deadlines. Our current work consists of a further investigation into the RL approach, and implementing the strategies in a real-life system.

## References

[1] R.J. Bril, C. Hentschel, E.F.M. Steffens, M. Gabrani, G.C. van Loo, and J.H.A. Gelissen. Multimedia QoS in consumer terminals. In *Proc. IEEE Workshop on Signal Processing Systems (SIPS)*, pages 332–343, Antwerp, Belgium, September 2001.

[2] K. Feng and A.K. Mok. A model of hierarchical real-time virtual resources. In *Proc. 23rd IEEE Real-Time Systems Symposium (RTSS)*, pages 26–35, Austin, TX, December 2002.

[3] C. Hentschel, R. Bril, M. Gabrani, L. Steffens, K. van Zon, and S. van Loo. Scalable video algorithms and dynamic resource management for consumer terminals. In *Proc. International Conference on Media Futures (ICMF)*, pages 193–196, Florence, Italy, May 2001.

[4] C. Lee, J. Lehoczky, R. Rajkumar, and D. Siewiorek. On Quality of Service optimization with discrete QoS options. In *Proc. Fifth IEEE Real-Time Technology and Applications Symposium (RTAS)*, pages 276–286, Vancouver, Canada, June 1999.

[5] C.W. Mercer, S. Savage, and H. Tokuda. Processor capacity reserves: Operating system support for multimedia applications. In *Proc. International Conference on Multimedia Computing and Systems (ICMCS)*, pages 90–99, Boston, MA, May 1994.

[6] S. Nataraja, editor. *Imprecise and Approximate Computation*. Kluwer Academic Publishers, Norwell, MA, 1995.

[7] C.M. Otero Pérez and I. Nitescu. Quality of service resource management for consumer terminals: Demonstrating the concepts. In *Proc. Work-in-Progress Session of the 14th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 29–32, Vienna, Austria, June 2002.

[8] M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Mathematical Statistics. Wiley-Interscience, New York, 1994.

[9] G. Slavenburg, S. Rathnam, and H. Dijkstra. The TriMedia TM-1 PCI VLIW mediaprocessor. In *Eight Symposium on High-Performance Chips, Hot Chips 8*, pages 171–177, Stanford, CA, August 1996.

[10] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[11] C.C. Wüst and W.F.J. Verhaegh. Quality control for scalable media processing applications. Accepted for publication in Journal of Scheduling.

[12] C.C. Wüst and W.F.J. Verhaegh. Dynamic control of scalable media processing applications. In E.H.L. Aarts, J.H.M. Korst, and W.F.J. Verhaegh, editors, *Algorithms in Ambient Intelligence*. Kluwer Academic Publishers, 2003. To appear.

# Dynamic Control of Scalable Media Processing Applications*

Clemens C. Wüst Wim F.J. Verhaegh

Philips Research Laboratories
Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands
{clemens.wust,wim.verhaegh}@philips.com

**Abstract**

## 1 Introduction

Consumer terminals, such as set-top boxes and digital TV-sets, become open and flexible by replacing hardware components that are dedicated to media processing by a programmable component on which equivalent media processing applications run. Media processing applications, such as MPEG-2 decoding, have two important properties. First, they show resource demands that vary significantly over time. This is due to the varying size and complexity of media data. Secondly, they have severe real-time demands to guarantee correct output. An ideal processing behavior is therefore obtained by assigning a media processing application a processing budget based on its worst-case load situation. However, to be cost-effective, resources should often be assigned closer to the average-case load situation. Without counter measures, this leads to the violation of real-time demands.
This problem can be managed by using scalable media processing applications. A scalable media processing application can run in different quality levels, leading to correspondingly different resource demands [5, 6, 7]. By controlling the quality level of processing, real-time demands can be satisfied, which contributes to a robust consumer terminal [3].
In this paper, we present an approach to dynamically control the quality level of processing during run time, based on actual processing time statistics. Our objective is to make an optimal trade-off between the quality of the output as perceived by the user and the required computational resources.

### 1.1 Related Work

### 1.2 Organization

This paper is organized as follows. A precise description of the problem and preliminary information are given in Section 2. Next, Section 3 describes the modeling of the problem as a finite Markov decision process. Using this model, in Section 4 we propose a quality level control strategy called scaled budget control, which is adaptive to structural load fluctuations. In Section 5 we discuss ... In Section 6 we test the performance of different control strategies by means of simulation experiments. Finally, in Section 7 we conclude the paper.

## 2 Problem Statement and Preliminaries

We consider a single scalable media processing application, shortly referred to as application. Its task is to continuously process units of work. Each unit of work is processed in one of the available quality levels,

*This research has been partially funded by ITEA through the EUROPA project [4].

and processed units of work are written to an output buffer. Units of work may mutually vary in size, type and complexity of processing. Hence, the time required to process a unit of work is considered to be stochastically distributed, dependent on both the unit of work type and the used quality level.
The moment in time at which the processing of a unit of work is finished, is called a *milestone*. For each milestone there is a deadline, and deadlines are assumed to be strictly *periodic in time*. A deadline corresponds to a moment in time at which some external event tries to extract a processed unit of work from the output buffer. Units of work and corresponding milestones and deadlines are numbered $1, 2, \ldots$. For convenience, we consider the moment in time right before processing the first unit of work also to be a milestone, numbered 0, although it has no corresponding deadline.
In each deadline period $P$, the application is guaranteed a fixed budget $b$ for processing. We define *relative progress* at a milestone as the total amount of budget left until the deadline of the milestone, divided by the periodic budget $b$. If the relative progress $\rho$ at a milestone drops below zero, so it was zero or larger at the previous milestone, then the application has encountered $\lceil -\rho \rceil$ deadline misses since the previous milestone. To deal with deadline misses, we consider two work preserving deadline miss approaches.
One approach is to use the output immediately upon creation, i.e., at the milestone at which a deadline miss is detected. This results in an adapted relative progress of 0 at the milestone. The deadlines for all next units of work are postponed by $-\rho P$ time units. We refer to this approach as the *conservative* deadline miss approach. Note that this approach presumes that deadlines can be shifted in time at a finer granularity than the deadline period.
Another approach is to use the output at the first next deadline. This results in an adapted relative progress of $\rho + \lceil -\rho \rceil$ at the milestone. The deadlines for all next units of work are postponed by $\lceil -\rho \rceil P$ time units. We refer to this approach as the *skipping* deadline miss approach.
Both deadline miss approaches result in a non-negative adapted relative progress at milestones. Therefore, a relative progress of 0 can be considered as a lower bound on relative progress. Further, we assume that the output buffer has a finite size. Therefore we also have an upper bound on relative progress, denoted by $\rho_{max}$.
At each milestone, a quality level is to be chosen at which the next unit of work is processed. The problem we consider is to choose this quality level in such a way that such that the following three objectives are met. First, the quality level at which units of work are processed should be as high as possible. Secondly, the number of deadline misses should be as low as possible, because deadline misses result in severe artefacts in the output. Thirdly, the number of quality level changes should also be as low as possible, because quality level changes also result in artefacts. A limiting condition is that a quality level control strategy runs on the same processor as the application, and therefore should take little computational resources.

## 3 Markov Policies

In this section, we model the problem of Section 2 as a finite Markov decision process (*MDP*). A finite Markov decision process describes a system that is characterized by a finite set of states, a finite set of decisions for each state, probabilities for state transitions which depend on the decision taken in a state, and revenues for taking a decision in a state. Solving the Markov decision process results in an optimal Markov policy. We use Markov policies to construct control strategies.

### 3.1 Model

**States.** At each milestone we consider the state of the application. This state is naturally given by the relative progress at the milestone. Because we need a finite set of states in the Markov decision process, we discretize relative progress into a finite set of equal-sized progress intervals between 0 and $\rho_{max}$. Further, to be able to compute revenues and transition probabilities later on, we extend the state with both the quality level at which the previous unit of work was processed and the type of the next unit of work.

**Decisions.** At each milestone a quality level is chosen at which the next unit of work is processed. Therefore, we model the set of decisions that can be taken in a state as the set of quality levels at which the application can run.

**Transition probabilities.** At milestones we have probabilities $p_{ij}^q$ for transitions from a state $i$ at the current milestone to a state $j$ at the next milestone, if quality level $q$ is chosen to process the next unit of work. To compute transition probabilities, we need processing time distributions for (unit of work type, quality level) pairs and probabilities for successive unit of work types. These statistical data can be obtained by processing a representative sequence of units of work in each available quality level. For the computation we further need to specify periodic budget $b$ and the applied deadline miss approach.
A problem in the computation is that the relative progress in state $i$, denoted by $\rho_i$, is required. However, not $\rho_i$ but only the corresponding progress interval is known, because this is part of state $i$. This problem is solved by approximating $\rho_i$ by the lower bound of the corresponding progress interval, which is a worst-case approximation. Clearly, the more progress intervals are chosen, the better the modeling of the transition probabilities becomes, as the approximated values of $\rho_i$ move closer to the real values.

**Revenues.** At milestones we have revenues $r_i^q$ for choosing a quality level $q$ in a state $i$. We use these revenues to implement the three problem objectives. First, the quality level at which units of work are processed should be as high as possible. We realize this by introducing a *utility function* $u(q)$, which returns a positive valued reward for choosing quality level $q$ at a milestone. This value should be directly related to the perceived output quality of the application, running at quality level $q$. Secondly, the number of deadline miss should be as low as possible. This is achieved by introducing a *deadline miss function*. Using budget $b$ and processing time distributions for (unit of work type, quality level) pairs, the expected number of deadline misses if quality level $q$ is chosen in state $i$ is computed. The deadline miss function returns this number multiplied by a positive valued deadline miss penalty. Finally, the number of quality level changes should be as low as possible. This is achieved by introducing a *quality change function* $qcp(q(i), q)$, which returns a positive valued penalty for changing from the previously used quality level, which is part of state $i$, to quality level $q$. A revenue $r_i^q$ is given by the utility function minus the deadline miss function and the quality change function.

### 3.2 MDP Instances

To define an MDP instance, a number of parameters have to be chosen. First, the relative progress upper bound $\rho_{max}$ and the number of progress intervals have to be chosen. As mentioned, the more progress intervals are chosen, the better the modeling of the problem becomes. Secondly, we have to define the utility function, the deadline miss function and the quality change function. These functions are the place to incorporate knowledge about user perception. Thirdly, to compute transition probabilities and revenues, we need statistical data. We derive these statistical data from a trace. A trace is a file which contains - for a specific application and a specific sequence of units of work - both the type and the processing times for different quality levels for the successive units of work. Finally, also periodic budget $b$ is to be chosen.
To solve an MDP instance, several Dynamic Programming related algorithms can be used, such as *successive approximation*, *value iteration* or *policy iteration*, see [8, 10, 11, 12]. A problem instance is solved off-line, and results in an optimal Markov policy. This Markov policy maps states at milestones to decision quality levels, independent of the time of the milestone.
Without loss of optimality, we can restrict ourselves to so-called *monotonic* Markov policies, i.e., per previously used quality level and next unit of work type we can assume that a higher relative progress results in a higher or equal quality level choice. Then, for storing a Markov policy, per previously used quality level only the relative progress bounds at which the policy changes from a particular quality level to the next one have to be stored. A Markov policy therefore has a space complexity of $O(|Q|^2)$, which is independent of the number of progress intervals. An Markov policy can be used directly to control the quality level of an application during run time.

## 4 Scaled Budget Control

In our earlier work [13], a simulation experiment showed that applying an optimal Markov policy to control the quality level of processing performs well if the processing times of successive units of work are not related. However, in practice this is often not the case. For example, Figure 1 shows the processing times for decoding a sequence of MPEG-2 frames (units of work) in one quality level. In this figure, both temporal

and structural load fluctuations are visible. The structural load fluctuations are due to the varying complexity of movie scenes.

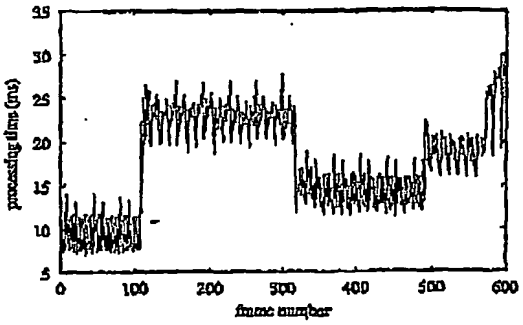



Figure 1. The load for decoding a sequence of MPEG-2 frames in one quality level, showing both temporal and structural fluctuations.

### 4.1 Load Complication

In Figure 1, the average load per frame over the entire sequence is about 17.5 ms. For subsequence frame $0-100$ the average load is about 10 ms, and for subsequence frame $100-300$ it is about 23 ms. If the sequence is processed using a budget $b = 17.5$ ms, then applying an optimal Markov policy derived for $b = 17.5$ ms would result in suboptimal quality level choices. In subsequence frame $0-100$, the Markov policy would be too pessimistic, leading to lower than necessary quality level choices. In subsequence frame $100-300$, the Markov policy would be too optimistic, leading to more than expected deadline misses.
One option to take the relation in load between successive units of work into account is by extending the state space of the Markov model with *load complication*. We define load complication as the proportion between the structural load and the average load for a given quality level of processing. We assume that the load complication is dependent on the complexity of the units of work that are processed, and not on the used quality level of processing. According to measurements for a scalable MPEG-2 decoder, this seems a fair assumption. Note that the load complication is also independent of the used budget.
The load complication in subsequence frame $0-100$ is about 0.57, and in subsequence frame $100-300$ it is about 1.31. To be included in the state space, load complication should just like relative progress be discretized. However, a problem is that the Markov model suffers from the *curse of dimensionality* [1]. This means that the number of states grows exponentially with the number of state variables. The state space would become too large for the problem to be solvable in reasonable time. Another drawback is that a lot more measure data would be required to derive reliable statistics for the problem. So, this approach to model the relation in load between successive units of work is undesirable.

### 4.2 Budget Scaling

Another approach is to model the relation in load between successive units of work is the following. Assume that the sequence is processed using a budget $b = 20$ ms. In subsequence frame $0-100$, the load complication is about 0.57. Clearly, processing at a load complication of 0.57 using budget $b$ is like processing at a load complication of 1, but using a budget that is 0.57 times smaller than its actual value. So, instead of applying an optimal Markov policy derived for $b = 20$ ms, we apply an optimal Markov policy derived for scaled budget $20ms/0.57 \approx 35$ ms. Similarly, in subsequence frame $100-300$ we apply an optimal Markov policy derived for scaled budget $20ms/1.31 \approx 15$ ms. We call this method to control the quality level of processing *scaled budget control*.
Clearly, it is not practical to derive optimal Markov policies for each possible scaled budget. Therefore, we only derive optimal Markov policies for a small number of budgets, and interpolate between these policies

to approximate an optimal Markov policy for a scaled budget.

### 4.3 Running Complication Factor

To compute the load complication during run-time, for each unit of work $i = 1, 2, \ldots$ we define *complication factor* $cf_i$ as follows. If unit of work $i$ is processed in quality level $q(i)$, yielding a processing time $pt(q(i))$, then its complication factor is given by

$$cf_i = \frac{pt_i(q(i))}{avg(q(i))}, \tag{1}$$

in which $avg(q(i))$ denotes the average processing time for quality level $q(i)$. Just like load complication we assume that the complication factor is independent of the used quality level. Clearly, the complication factor varies per unit of work, following the temporal and the structural load fluctuations. To make it follow only the structural load fluctuations, we weaken the temporal load fluctuations by computing a running average over the complication factors, called *running complication factor*. To compute a running average we use a technique called *exponential recency weighted average* [10]. Using an initial estimate $rcf_0 = 1$, the running complication factor for unit of work $i$ is given by

$$rcf_i = rcf_{i-1} + \alpha * (cf_i - rcf_{i-1}), \tag{2}$$

in which $\alpha$, $0 < \alpha \leq 1$ is a constant step-size parameter. The sum of the weights over all processing times $pt_j(q)$ for $j = 1, \ldots, i$ is 1. Each processing time $pt_j(q)$, $j \leq i$, is given a weight $\alpha \cdot (1-\alpha)^{i-j}$, which decreases exponentially as $i$ increases. The closer $\alpha$ is to zero, the more farsighted the running average becomes. In case $\alpha = 1$, only the last encountered processing time contributes to the running average. By carefully selecting $\alpha$, the running complication factor approximates the load complication.

### 4.4 Trace Normalization

We have to revisit the statistics that we use to derive optimal Markov policies. Because we already take care of structural load fluctuations by means of scaling the budget that is used to select an optimal Markov policy, we want to derive optimal Markov policies using statistics based only on temporal load fluctuations. For this purpose we use a normalized trace, comprising only temporal load fluctuations, which is created as follows.

Given a trace, we first compute for each quality level $q$ the average processing times over all units of work, denoted by $avg(q)$. These values for $avg(q)$ are also used in (1). Next, by running through the trace, we compute for each quality level $q$ a running complication factor $rcf_i(q)$ over the processing times of all units of work $i$ encountered so far. Using initial estimates $rcf_0(q) = 1$, the running complication factor is given by

$$rcf_i(q) = rcf_{i-1}(q) + \alpha * \left(\frac{pt_i(q)}{avg(q)} - rcf_{i-1}(q)\right), \tag{3}$$

using the same value for $\alpha$ as used in (2). A normalized trace is created by dividing each processing time $pt_i(q)$ by $rcf_{i-1}(q)$.

## 5 Off-line Optimum

## 6 Simulation Experiments

To test our model, we have run simulation experiments. These experiments are based on the statistics produced by a scalable MPEG-2 decoder running on a *TriMedia* processor [9]. Given an MPEG-2 coded video stream as input, the decoder sequentially decodes the frames of which the stream is made up. We let frames correspond with units of work. Frames can independent of each other be decoded in five different quality levels, in increasing quality order named $q_1, \ldots, q_5$. There are three different types of MPEG-2

frames: intra coded (I-frames), predictive coded (P-frames), and bidirectional coded (B-frames). The time required to decode a frame of a given type in a given quality level is variable, mainly dependent on the complexity of the frame.
We use the scalable MPEG-2 decoder to create traces for MPEG-2 coded video streams. A trace contains for each frame in the video stream its type and the time required to decode it in each quality level $q_1, \ldots, q_5$. Traces serve two purposes. First, to derive statistics that are used to compute transition probabilities and revenues for MDP instances. Second, to simulate the execution of the decoder, which we do to test quality level control strategies.
We define the parameters for MDP instances as follows. The upper bound on relative progress $p_{max}$ is chosen equal to 3, which corresponds to using output buffer in which 3 decoded frames can be stored. We choose 500 progress intervals inbetween a relative progress of zero and $p_{max}$. This number of progress intervals is the result of a trade-off made between the solution quality and the computational complexity of the MDP instances, for which we refer to [13]. The utility function $u(q)$ is defined by $u(q_1) = 1$, $u(q_2) = 2.5$, $u(q_3) = 5$, $u(q_4) = 7.5$, $u(q_5) = 10$. The deadline miss penalty is defined by 1000, which roughly means that we allow at most 1 deadline miss per 100 frames. The quality change function is defined by a penalty of 0 for not changing the quality level at a milestone, a penalty of 4 for moving 1 quality level up or down, a penalty of 7 for moving 2 quality levels up or down, a penalty of 9 for moving 3 quality levels up or down, and a penalty of 10 for moving 4 quality levels up or down.
Apart from these fixed parameters, we also have three parameters that we vary to define different MDP instances. First, we vary the trace whose statistics are used to compute transition probabilities and revenues. Secondly, we can choose whether or not frame types should be taken into account. If frame types are not taken into account, I-,P- and B-frames are all regarded to be of the same type. To distinguish between both cases, we attach a subscript *ft* to the name of the trace to indicate that frame types are taken into account. Finally, we also vary budget $b$ to define different MDP instances.
Given a trace and a decision about the use of frame types, we define a set of 41 different MDP instances by varying $b$ from 5 ms to 45 ms, using intermediate steps of 1 ms. To solve the set of MDP instances we apply successive approximation using an inaccuracy parameter of 0.01. This results in a set of optimal Markov policies, one policy for each budget. Next, we use the policies to construct quality level control strategies. If the decoder processes using a fixed budget $b$, then we can directly apply the optimal Markov policy derived for $b$ to control the quality level at which frames are decoded. For trace $\tau$ this control strategy is denoted by MP($\tau$). We can also use the set of optimal Markov policies to implement scaled budget control. For trace $\tau$ and step-size parameter $\alpha$ this control strategy is denoted by SBC($\tau$,$\alpha$).
To test a control strategy, we simulate the processing of the decoder using the control strategy 41 times, for $b$ varying from 5 ms to 45 ms using intermediate steps of 1 ms. In these simulation we apply the skipping deadline miss approach. To compare control strategies, in each simulation we measure the quality level usage, the deadline misses and the quality level changes. From these measurements we compute for each simulation the *average revenue per transition*, which is the thing we want to optimize.
We have performed three experiments.

### 6.1 Performance of Control Strategies

### 6.2 The influence of frame types

### 6.3 The influence of the trace

## 7 Conclusions

later invullen

## References

[1] R.E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, 1957.

[2] R.J. Bril, M. Gabrani, C. Hentschel, G.C. van Loo, and E.F.M. Steffens. QoS for consumer terminals

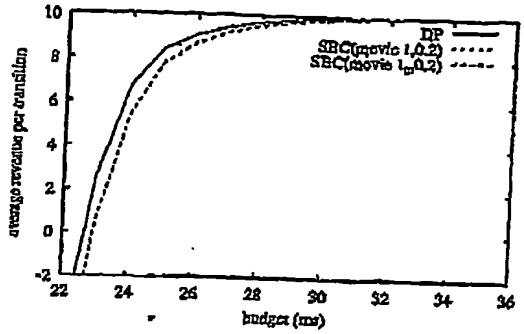



Figure 2.

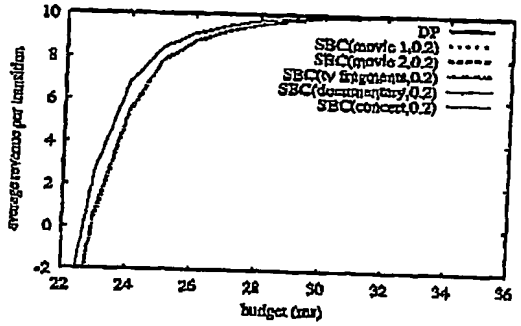



Figure 3.

and its support for product families. In *Proc. International Conference on Media Futures (ICMF)*, pages 299–302, Florence, Italy, May 2001.

[3] R.J. Bril, C. Hentschel, E.F.M. Steffens, M. Gabrani, G.C. van Loo, and J.H.A. Gelissen. Multimedia QoS in consumer terminals. In *Proc. IEEE Workshop on Signal Processing Systems (SIPS)*, Antwerp, Belgium, September 2001.

[4] J.H.A. Gelissen. The ITEA project EUROPA, a software platform for digital CE appliances. In *Proc. International Conference Media Futures (ICMF)*, pages 157–160, Florence, Italy, May 2001.

[5] C. Hentschel, R. Bril, M. Gabrani, L. Steffens, K. van Zon, and S. van Loo. Scalable video algorithms and dynamic resource management for consumer terminals. In *Proc. International Conference on Media Futures (ICMF)*, pages 193–196, Florence, Italy, May 2001.

[6] T. Lan, Y. Chen, and Z. Zhong. MPEG2 decoding complexity prediction and dynamic complexity control in a Trimedia processor. In *Proc. Workshop on Multimedia Signal Processing*, Cannes, France, October 2001.

[7] S. Peng. Complexity scalable video decoding via IDCT data pruning. In *Digest of Technical Papers IEEE International Conference on Consumer Electronics (ICCE)*, pages 74–75, June 2001.

[8] M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons Inc., 1994.

[9] G. Slavenburg, S. Rathnam, and H. Dijkstra. The Trimedia TM-1 PCI VLIW mediaprocessor. In *Eigth Symposium on High-Performance Chips, Hot Chips 8*, pages 171–177, Palo Alto, California, 1996.

[10] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[11] J. van der Wal. *Stochastic Dynamic Programming*. PhD thesis, Mathematisch Centrum Amsterdam, 1980.

[12] D.J. White. *Markov Decision Processes*. John Wiley & Sons Ltd., 1993.

[13] C.C. Wüst and W.F.J. Verhaegh. Quality level control for scalable media processing applications. Submitted to: Journal of Scheduling.

CLAIMS

1. Method of setting an output quality of a next media frame, wherein the output quality is provided by a media processing application; and the media processing application is designed to provide a plurality of output qualities of the next media frame
characterized in that setting the output quality of the next media frame is based upon a self-learning control strategy that uses a processing time and an output quality of a previous media frame to determine the output quality of the next media frame.

2. System to set an output quality of a next media frame, comprising application means conceived to provide the output quality of a plurality of output qualities of the next media frame
characterized in that the system further comprises
control means conceived to set the output quality of the next media frame based upon a self-learning control strategy that uses a processing time and an output quality of a previous media frame to determine the output quality of the next media frame.

3. A computer program product designed to perform the method according to claims 1.

4. A storage device comprising a computer program product according to claim 3.

5. A television set comprising a system according claim 2.

6. A television set comprising a system according claim 2.